Appendix A

Structure *from Data*

The top-down (qualitative, structural) and bottom-up (quantitative, numeric) modeling approaches are fundamentally incompatible[1]. This report concentrates on the data-oriented modeling approach: The discussions were based exclusively on data. However, here in these Appendices we try to bridge the gap between the two extremes — or, at least, we try to bring the two approaches nearer to each other.

This first appendix concentrates on the approach *from data towards structure,* that is, it is assumed that analysis of data suggests some underlying structure explaining the observations. The data-suggested structure is typically seen as the clustered nature of the data. The latter appendix concentrates on the approach *from structure towards data,* that is, the data is explicitly modified to match the known structure. In both cases, these structure-oriented analyses are carried out *before* the actual data modeling, or regression analysis, is done.

A.1 Cluster analysis

Determination of the system structure is, a complex and knowledge-intensive task. Typically, when doing multivariate modeling, no *a priori* information about the underlying subprocesses exists. This complexity is reflected in the data: The emergence of relevant clusters cannot be foreseen. There exist no unique solutions to the data clustering problem, and clustering is typically based on more or less heuristic algorithms. Because of the nonlinear and noncontinuous nature of the clustering problem, there exist only iterative, trial-and-error algorithms for this purpose.

In what follows, two prototypical clustering approaches will be studied a little closer. Both of them work well only for "nice" data, hoping that the clusters are more or less clearly distinguishable in the measurements.

[1] It is as in artificial intelligence (AI) — either you do it symbolically with expert systems, etc., or you do it numerically with neural networks, etc. — and there seem not to exist natural combinations in between

A.1.1 K-means algorithm

The *K-means algorithm* is the basic approach that is used for clustering: Starting from some initial guesses, the clusters are refined by moving samples from a cluster to another depending on which of the clusters happens to be closest. When samples are redistributed, the cluster centers also evolve; iteration is needed to reach convergence.

The algorithm that searches for N distinct clusters (parameter N being fixed beforehand) can be written as follows (references to the clusters are now shown as superscript indices):

1. Choose a set of original cluster centers $\bar{v}^1, \ldots, \bar{v}^N$ arbitrarily, for example, let $\bar{v}^1 = v(1), \ldots, \bar{v}^N = v(N)$.

2. Assign the k samples to the N clusters using the minimum Euclidean distance rule: Sample $v(\kappa)$ belongs to cluster c, or $v(\kappa) \in \Gamma^c$, if $\|v(\kappa) - \bar{v}^c\| \leq \|v(\kappa) - \bar{v}^{c'}\|$ for all $c' \neq c$.

3. Compute new cluster center prototypes $\bar{v}^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} v(\kappa)/\#\{\Gamma^c\}$, where $\#\{\Gamma^c\}$ denotes the number of vectors in cluster Γ^c.

4. If any of the cluster prototypes changes, return to step 2, otherwise, stop.

Note that it is assumed that v contains now all available information, containing both the input variables (later denoted x) and the output variables (later y); in some sources this approach is called "input-output clustering". It is also possible to use x exclusively[2].

If one determines some *topology* (indeed, a *metric*) among the clusters, so that some of the clusters are assumed to be "nearer" to each other than some others, one can easily extend the K-means algorithm towards *self-organizing map*. If it is not only the cluster itself whose center is moved towards the local data center, but also its "neighbors" are slightly adapted in the same direction, one has (a version of) the *Batch-SOM* algorithm [**?**]. In the converged cluster organization, the "neighboring" clusters will stand for nearby data samples, so that a "map" is constructed.

The K-means clustering method works reliably, and sometimes it gives useful results. However, there is a basic problem: The distances are calculated using the Euclidean norm. If searching for linear dependency models within the clusters, it is not pointwise but linear, "longish" data clusters that support linear model construction. How this can be achieved, is studied next.

A.1.2 EM algorithm

The *Expectation Maximization (EM)* algorithm is a more sophisticated approach as compared to the basic K-means algorithm: Clustering is serched for in the maximum likelihood sense, fitting Gaussian distributions with data in a more

[2]Indeed, this is the normal approach: When the models are applied, it is only the input x that is available for determining the cluster

complicated way ... needless to say that there is no guarantee about the convergence or the uniqueness of the solutions. It is not only the cluster centers, or means of the distributions, but also the "outlooks", or covariance matrices, that need to be determined here; this means that the number of free modl parameters is very high. As there exist various local minima, bootstrapping the algorithm is also complicated — typically the initial guesses for clusters are calculated using the K-means algorithm.

First, study the Gaussian distribution (2.1) a bit closer. Probability density reaches maximum simultaneously as its (natural) logarithm does; this means that one can define the *log-likelihood measure* for each cluster:

$$\begin{aligned} \ln(p(v)) \quad = \quad & -\tfrac{\dim\{v\}}{2} \cdot \ln(2\pi) \\ & -\tfrac{1}{2} \cdot \ln(\det\{R^c\}) \\ & -\tfrac{1}{2} \cdot (v-\bar{v}^c)^T (R^c)^{-1} (v-\bar{v}^c). \end{aligned} \tag{A.1}$$

This criterion can be applied for determining into which cluster a data sample should be put to maximize the overall model fit with the data. The first term above is constant for different clusters, and it can be neglected; the role of the second term, regulating the *a priori* cluster probability to fixed level, is to prevent the cluster from growing unboundedly. Finally, the third term matches the data points against the Gaussian models within clusters; essentially one calculates the *Mahalanobis distance* from the data point to the cluster c:

$$(v-\bar{v}^c)^T (R^c)^{-1} (v-\bar{v}^c). \tag{A.2}$$

This measure determines how "longish" the distribution is; searching for the locations of the equidistant points in the v space using this distance measure, ellipsoids are found. Based on log-likelihood, the EM algorithm can be written as

1. Choose a set of original cluster centers $\bar{v}^1, \ldots, \bar{v}^N$ arbitrarily, for example, using the K-means algorithm; the cluster covariances are originally identity matrices, or $R^c = I$.
2. Assign the k samples to the N clusters using the minimum (balanced) Mahalanobis distance rule: Sample $v(\kappa)$ belongs to cluster c, or $v(\kappa) \in \Gamma^c$, if $\ln(\det\{R^c\}) + (v(\kappa)-\bar{v}^c)^T (R^c)^{-1} (v(\kappa)-\bar{v}^c)$ becomes minimum.
3. Compute new cluster center prototypes $\bar{v}^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} v(\kappa)/\#\{\Gamma^c\}$ and covariance estimates $R^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} (v(\kappa)-\bar{v}^c)(v(\kappa)-\bar{v}^c)^T/\#\{\Gamma^c\}$ where $\#\{\Gamma^c\}$ denotes the number of vectors in cluster Γ^c.
4. If any of the cluster prototypes changes, return to step 2, otherwise, stop.

Note that K-means algorithm results if it is explicitly assumed that in all clusters $R^c \equiv \sigma^2 \cdot I$ for some σ^2. The EM algorithm can be made more stable if some additional assumptions can be made. For example, if one can assume that the nonlinearity within the data is simple *affinity,* so that only the cluster centers vary while the internal structures within the clusters remains unchanged, there holds $R^c = R^{c'}$ for all c and c' — this effectively reduces the problem complexity.

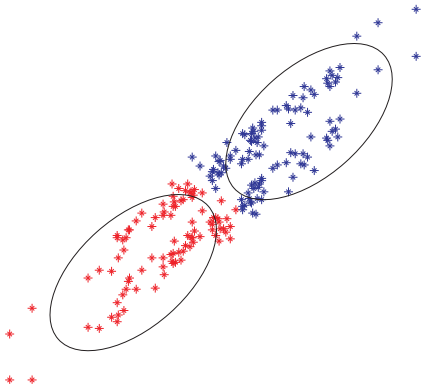

Figure A.1: Incorrect, K-means type clustering result

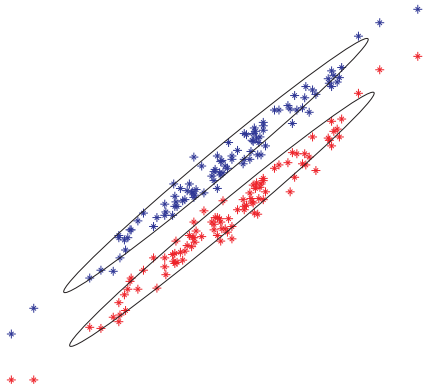

Figure A.2: Intuitively correct clustering result

The presented EM algorithm matches well with the (Gaussian) mixture model scheme that was discussed in Chapter 2: Data samples within the clusters have the direct probability interpretation, so that the combination of the submodels constructed for individual clusters can be carried out in the maximum likelihood sense.

A.2 Classification

Sometimes the appropriate classes are already known — but they are known only by examples. Then one is facing a *classification problem*: How to determine the *decision boundary* between the classes, so that, when facing fresh data, the probability of false classifications would be minimized?

A.2.1 Fisher discriminant analysis

Knowing the clusters, it would be good to know some structure among the clusters. Additionally, sometimes it would be nice to have a *linear* criterion for determining how well a new sample matches a cluster and how near it is to the neighboring clusters. One would like to find a projection axis so that data belonging to different clusters, as projected onto this axis, would be maximally distinguishable.

From the classification point of view, one can assume that the clusters carry the classification information, whereas the variation around the cluster centers can be interpreted as noise. One can try to find such a projection of the data that the *signal-to-noise ratio* is maximized. So, first define the "noise sequence" so that the cluster centers $\bar{v}^{c(\kappa)}$ corresponding to each of the classified sample $v(\kappa)$ is eliminated:

$$v_{\text{noise}}(\kappa) = v(\kappa) - \bar{v}^{c(\kappa)}. \tag{A.3}$$

The signal sequence, then, is the sequence of cluster centers:

$$v_{\text{signal}}(\kappa) = \bar{v}^{c(\kappa)}. \tag{A.4}$$

Assume that the projection axis that is being searched for is θ_i. A data sample $v(\kappa)$ projected onto this axis is $v^T(\kappa)\theta_i$; its square thus is $\theta_i^T v(\kappa)v^T(\kappa)\theta_i$. The average of this, or the variance, can be written separately for the signal and the noise sequences, giving

$$\begin{aligned} &\tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} \theta_i^T v_{\text{signal}}(\kappa) v_{\text{signal}}^T(\kappa)\theta_i \\ &\quad = \theta_i^T \cdot \tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} v_{\text{signal}}(\kappa) v_{\text{signal}}^T(\kappa) \cdot \theta_i \\ &\quad = \theta_i^T \cdot R_{\text{between}} \cdot \theta_i, \end{aligned} \tag{A.5}$$

and

$$\begin{aligned} &\tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} \theta_i^T v_{\text{noise}}(\kappa) v_{\text{noise}}^T(\kappa)\theta_i \\ &\quad = \theta_i^T \cdot \tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} v_{\text{noise}}(\kappa) v_{\text{noise}}^T(\kappa) \cdot \theta_i \\ &\quad = \theta_i^T \cdot R_{\text{within}} \cdot \theta_i. \end{aligned} \tag{A.6}$$

Here, the matrices R_{between} and R_{within} denote the "between-classes" covariance and the "within-classes" covariance, respectively. Now the problem of maximizing the between-classes variance while keeping the within-classes variances constant can be expressed as a constrained optimization task

$$\begin{array}{ll} \text{Maximize} & \theta_i^T \cdot R_{\text{between}} \cdot \theta_i \\ \text{when} & \theta_i^T \cdot R_{\text{within}} \cdot \theta_i = 1. \end{array} \tag{A.7}$$

This can be formulated in the Lagrangian framework (see page 20) when selecting

$$\left\{ \begin{array}{lcl} f(\theta_i) & = & \theta_i^T \cdot R_{\text{between}} \cdot \theta_i \\ g(\theta_i) & = & 1 - \theta_i^T \cdot R_{\text{within}} \cdot \theta_i. \end{array} \right. \tag{A.8}$$

Using the Lagrange multipliers, the optimum solution θ_i has to obey

$$\frac{d\,J(\theta_i)}{d\theta_i} = \frac{d}{d\theta_i}\left(f(\theta_i) - \lambda_i \cdot g(\theta_i)\right) = \mathbf{0} \tag{A.9}$$

or

$$R_{\text{between}} \cdot \theta_i - \lambda_i \cdot R_{\text{within}} \cdot \theta_i = \mathbf{0}, \tag{A.10}$$

giving

$$R_{\text{between}} \cdot \theta_i = \lambda_i \cdot R_{\text{within}} \cdot \theta_i. \tag{A.11}$$

If the matrix R_{within} is invertible, this can further be solved as

$$R_{\text{within}}^{-1} R_{\text{between}} \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{A.12}$$

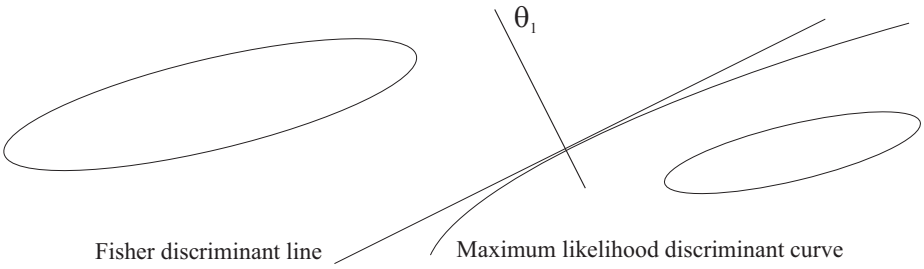

Figure A.3: Fisher discriminant axis θ_1 in a two-cluster case with unequal covariances. Note two things: First, the discriminant axis is not pointed from one cluster center to the other, the covariance structure of the clusters affecting its orientation; second, the maximum likelihood discriminant surface between clusters is generally not a plane but an hyperellipsoid (or some other generalized conic section determined by the equi-distance points in the Mahalanobis sense)

This is an *eigenproblem* (and (A.11) is so called *generalized eigenproblem*)[3]. That is, the best projection axes are given as eigenvectors of the above problem. The eigenvector corresponding to the largest eigenvalue is the best in this sense. Note that if there are only two clusters, the axis is unique (the rank of R_{between} being 1, all but one of the eigenvalues being zeros).

As an example, assume that there are only two clusters with equal covariances. In this case the *discriminant (hyper)plane* between the clusters is defined by those points that lie on the hyperplane going through the center point between the clusters and being perpendicular to the axis θ_1. However, if the clusters do *not* have equal covariance structures, the linear Fisher discriminant no longer gives the theoretically correct separation between the clusters (see Fig. A.3).

If one is trying to find an appropriate way of scaling ones data, the FDA model can also give some intuition. The signal-to-noise ratio information can directly be used for weighting purposes according to the weighting scheme (B.31). This approach is generally used, more or less knowingly, for example, in *data mining:* The relevance of different words in textual documents is estimated by checking how often they are found in "interesting" documents and, on the other hand, in "non-interesting" ones. This information about frequency variations can be used for weighting the words appropriately.

A.2.2 Support Vector Machines (SVM)

One of the most promising modern classification methods is called a *Support Vector Machine* or SVM for short [**?**]. It is a result of sophisticated mathematics, optimization, and statistical learning theory — and, surprisingly, the basic ideas are very well compatible with the ideas of multivariate regression as studied in this report:

- The structural complexity of the decision boundaries is substituted with

[3]Later we will see how often the same problem formulation pops up in multivariate analysis

dimensional complexity, that is, the original data space is augmented with a high number of feature variables.

- In the high-dimensional feature space, it is assumed that the patterns are linearly separable from each other, and very efficient linear classification methods are applied.
- The structural information about the samples can be expressed as *kernel matrices,* representing "similarities", being closely connected to association matrices that were studied before.

There are also some differences in interpretations. First, in pattern recognition applications it is assumed that the number of features is huge — typically the number of samples is lower than the number of features, $k \leq n$. Because of this, the kernel matrices, for example, are calculated "horizontally" for the *sample vectors.* Note that the covariance properties remain here essentially the same, only the matrix dimensions become lower.

It is also the objective — classification rather than regression — that means that there are some complications. After all, the adaptation is necessarily nonlinear and iterative: It is only those samples ("support vectors") that are located nearest to the decision boundary that are of essence in classifier training, the other samples are automatically correctly classified. The SVM algorithm maximizes the error margin; it maximizes the minimum distance between the support vectors and the separating hyperplane.

The SVM's are not concentrated on here in more detail. Perhaps it suffices to say that — despite its mathematically simple and elegant ideas, it often outperforms more sophisticated nonlinear approaches.

Computer exercises

1. You can test the behaviors of different clustering algorithms by using the analysis and data construction routines in `Regression Toolbox` for `Matlab`. For example, you can try the following commands:

```
DATA = dataClust(3,5,50,20,100);   % See "help dataclust"
clustersKM = regrKM(DATA,5);
regrShowClust(DATA,clustersKM);
clustersEM = regrEM(DATA,5);
regrShowClust(DATA,clustersEM);
```

2. Extend the K-means algorithm `regrKM` in the `Regression Toolbox` so that it approximately implements the *Batch-SOM* algorithm. For simplicity, you can restrict to *one-dimensional* maps, so that the clusters $c-1$ and $c+1$ are the nearest neighbors of the cluster number c.